SHANDONG UNIVERSITY

More Inputs Makes Difference: Implementations of Linear Layers Using Gates with More Than Two Inputs

Qun Liu[1,2] Weijia Wang[1,2] Ling Sun[1,2] Yanhong Fan[1,2]
Lixuan Wu[1,2] Meiqin Wang(✉)[1,2,3]

[1]Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China

[2]School of Cyber Science and Technology, Shandong University, Qingdao, China

[3]Quan Cheng Shandong Laboratory, Jinan, China

March 22, 2023

Outline

Outline

1. Background
2. Motivation and Contribution
3. New Aspects for SLP Problem
4. Graph Extending Algorithm
5. Transforming Framework
6. Application

Lightweight Cryptography

Applications

- Internet of Things
- Radio-Frequency Identification tags

Limitations

- The circuit size
- The power consumption
- The latency

Directions

- Designing Lightweight Primitives
- Optimizing Existing Implementations

Lightweight Cryptography

Applications

- Internet of Things
- Radio-Frequency Identification tags

Limitations

- The circuit size
- The power consumption
- The latency

Directions

- Designing Lightweight Primitives
- Optimizing Existing Implementations

Lightweight Cryptography

Applications

- Internet of Things
- Radio-Frequency Identification tags

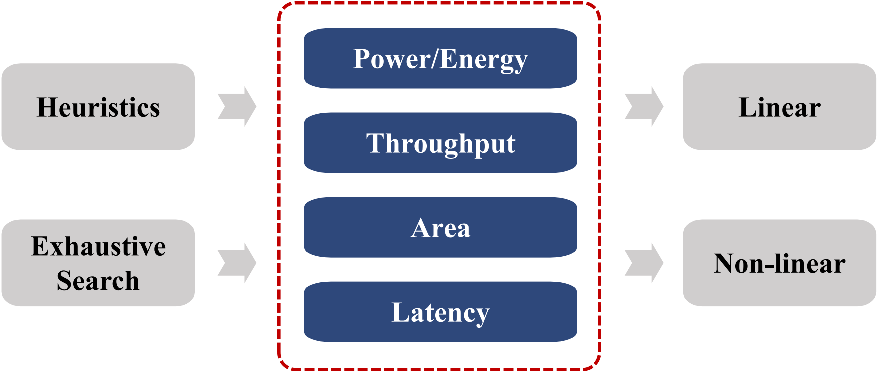
Limitations

- The circuit size
- The power consumption
- The latency

Directions

- Designing Lightweight Primitives
- Optimizing Existing Implementations

Criteria

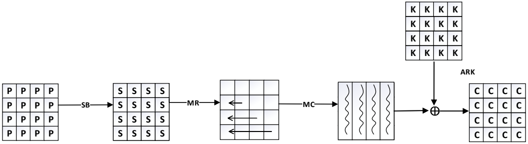


Our work

In this paper, our work mainly focuses on the area of linear layers.

Outline

Motivation and Contribution



AES round function

An example

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

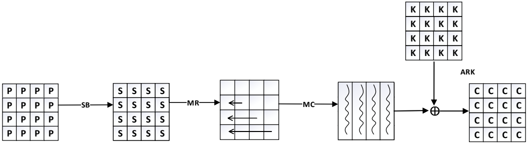
$$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_2 = x_3 \oplus x_4$$

8 XOR gates -> 16 GE

$$y_2 = x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus y_2$$
$$y_0 = x_0 \oplus y_1$$

4 XOR gates -> 8 GE

Motivation and Contribution



AES round function

An example

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

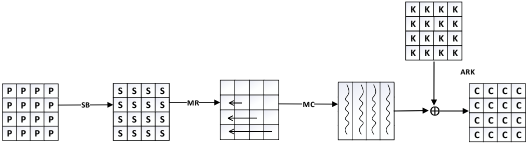
$$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_2 = x_3 \oplus x_4$$

8 XOR gates -> 16 GE

$$y_2 = x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus y_2$$
$$y_0 = x_0 \oplus y_1$$

4 XOR gates -> 8 GE

Motivation and Contribution



AES round function

An example

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_2 = x_3 \oplus x_4$$

8 XOR gates -> 16 GE

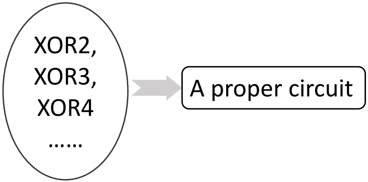
$$y_2 = x_3 \oplus x_4$$
$$y_1 = x_1 \oplus x_2 \oplus y_2$$
$$y_0 = x_0 \oplus y_1$$

4 XOR gates -> 8 GE

Motivation and Contribution

Previous Work

- Paar's work (first work)
- Boyar *et al.* (efficient algorithm)
- Siwei Sun *et al.* (depth limitation)
- Quanquan Tan *et al.*, Zejun Xiang *et al.*, $\cdots$



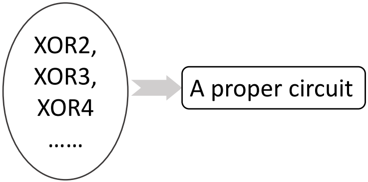
Further Work – Multi-Input Gates

- Directly search. (Baksi *et al.*) (Too large search space)
- Transform strategy. (Banik *et al.*) (Requiring more candidates)

Motivation and Contribution

Previous Work

- Paar's work (first work)
- Boyar *et al.* (efficient algorithm)
- Siwei Sun *et al.* (depth limitation)
- Quanquan Tan *et al.*, Zejun Xiang *et al.*, $\cdots$



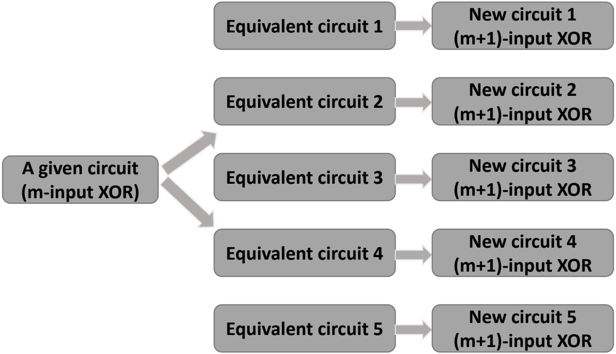
Further Work – Multi-Input Gates

- Directly search. (Baksi *et al.*) (Too large search space)
- Transform strategy. (Banik *et al.*) (Requiring more candidates)

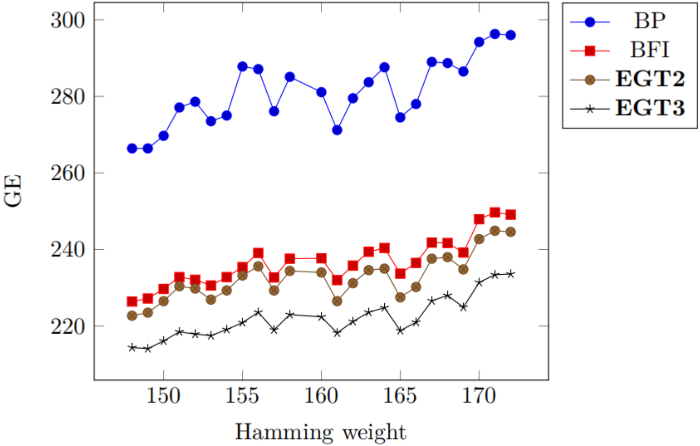
Motivation and Contribution

Contribution

- The transforming framework (n to $n+1$)
- The graph extending algorithm
- Application to many linear layers of block ciphers (2/3/4-input XOR gates)



Experiment: 5500 matrices



Outline

New Aspects for SLP Problem

SLP Problem

The Shortest Linear Program (SLP) problem is defined as finding a solution with the minimum number of XORs to compute the multiplication of an $m \times n$ constant matrix A over $\mathbb{F}_2$.

Finding Solutions

It's NP-hard, and no efficient algorithm can solve it exactly.

- Optimization
- Computational geometry
- Operations research

New Aspects for SLP Problem

SLP Problem

The Shortest Linear Program (SLP) problem is defined as finding a solution with the minimum number of XORs to compute the multiplication of an $m \times n$ constant matrix A over $\mathbb{F}_2$.

Finding Solutions

It's NP-hard, and no efficient algorithm can solve it exactly.

- Optimization
- Computational geometry
- Operations research

New Aspects for SLP Problem

New Problem

The minimum XORs vs. the lowest area

SLPA Problem (SLP problem with the lowest Area)

Given the cost λ_i $(1 \leq i \leq \epsilon)$ of every operation, the metric is defined as

$$\min(\lambda_1 e_1 + \lambda_2 e_2 + \ldots + \lambda_\epsilon e_\epsilon),$$

where e_i counts the number of the i-operation.

ϵ-operation $(\epsilon \in \mathbb{N})$: an operation containing ϵ 2-input xor gates.
1-operation: XOR2, 2-operation: XOR3, 3-operation: XOR4

Directed Acyclic Graph

Directed Acyclic Graph (DAG)

A directed acyclic graph is a directed graph that has no cycles.

Topological Ordering

The topological ordering T_G of a directed acyclic graph G is an ordering of its nodes into a sequence.

Directed Acyclic Graph

Directed Acyclic Graph (DAG)

A directed acyclic graph is a directed graph that has no cycles.

Topological Ordering

The topological ordering T_G of a directed acyclic graph G is an ordering of its nodes into a sequence.

Outline

An Example

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$t_{8,0,1}, t_{9,2,3}, t_{10,4,5}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, t_{\underline{14},4,11}, t_{\underline{15},5,11}, t_{\underline{16},12,13}, t_{\underline{17},13,14},$

It requires 10 XOR gates.

Single Graph and Extended Graph

Single Graph and Extended Graph

The single graph is a directed graph so that each non-unit node has only one implementation.
The extended graph is the directed graph so that each node can have more than one implementation.

$$t_{8,0,1},\ t_{9,2,3},\ t_{10,4,5},\ t_{11,6,7},\ t_{\underline{12},8,9},\ t_{13,9,10},\ t_{\underline{14},4,11},\ t_{\underline{15},5,11},\ t_{\underline{16},12,13},\ t_{\underline{17},13,14}. \tag{1}$$

$$\begin{aligned}&t_{8,0,1},\ t_{9,2,3},\ \{t_{10,4,5},\ t_{10,14,15}\},\ t_{11,6,7},\ t_{\underline{12},8,9},\ t_{13,9,10},\ \{t_{\underline{14},4,11},\ t_{\underline{14},10,15}\},\\ &\{t_{\underline{15},5,11},\ t_{\underline{15},10,14}\},\ \{t_{\underline{16},12,13},\ t_{\underline{16},8,10}\},\ \{t_{\underline{17},13,14},\ t_{\underline{17},9,15}\}.\end{aligned} \tag{2}$$

Generating the Extended Graph

Algorithm 1 GenerateExtendedGraph()

Input: A *single graph* G_s and the operation *op* (2 or 3)
Output: An *extended graph* G_e
R_{G_s} = GetReachabilitySet(G_s)
if TopologicalOrdering(G_s) = *error* **then** ▷ Checking the cycles
  return *error*
end if
T = TopologicalOrdering(G_s)
$G_e \leftarrow G_s$
for i from 1 to $|T| - 1$ **do** ▷ Checking whether two nodes has the same value
  $u = T[i]$
  for j from $i+1$ to $|T|$ **do**
    $v = T[j]$
    if $u = v$ **then**
      Remove v and let the origin of each edge whose origin is v be u
    end if
  end for
end for
for each $u \in G_s$ **do** ▷ Generating the *extended graph* G_e
  if u is not unit node **then**
    $\mathcal{A} \leftarrow \phi$ ▷ The *available set* of all the nodes
    for each $v \in G_s/\{u\}$ **do**
      if v not in R_u **then**
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{v\}$
      end if
    end for
    if ($|\mathcal{A}| < 2$ and $op = 2$) or ($|\mathcal{A}| < 3$ and $op = 3$) **then**
      Continue
    end if
    if $op = 2$ **then**
      for $w, v \in \mathcal{A}(w \neq v)$ **do** ▷ Using XOR2
        if $u = w \oplus v$ and u has not the implementation (w, v) **then**
          Add (w, v) for u in G_e ▷ Adding a new implementation for u
        end if
      end for
    end if
    if $op = 3$ **then**
      for $w, v, p \in \mathcal{A}(w \neq v \neq p)$ **do** ▷ Using XOR3
        if $u = w \oplus v \oplus p$ and u has not the implementation (w, v, p) **then**
          Add (w, v, p) for u in G_e ▷ Adding a new implementation for u
        end if
      end for
    end if
  end if
end for

Nodes	reachability sets
t_0, t_1	$\{t_8, t_{12}, t_{16}\}$
t_2, t_3	$\{t_9, t_{12}, t_{13}, t_{16}, t_{17}\}$
t_4	$\{t_{10}, t_{13}, t_{14}, t_{16}, t_{17}\}$
t_5	$\{t_{10}, t_{13}, t_{15}, t_{16}, t_{17}\}$
t_6, t_7	$\{t_{11}, t_{14}, t_{15}, t_{17}\}$
t_8	$\{t_{12}, t_{16}\}$
t_9	$\{t_{12}, t_{13}, t_{16}, t_{17}\}$
t_{10}	$\{t_{13}, t_{16}, t_{17}\}$
t_{11}	$\{t_{14}, t_{15}, t_{17}\}$
t_{12}	$\{t_{16}\}$
t_{13}	$\{t_{16}, t_{17}\}$
t_{14}	$\{t_{17}\}$
t_{15}, t_{16}, t_{17}	ϕ

Splitting the Extended Graph

$t_{8,0,1}, t_{9,2,3}, t_{10,4,5}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, t_{\underline{14},4,11}, t_{\underline{15},5,11}, t_{\underline{16},12,13}, t_{\underline{17},13,14}.$

The single graph

$t_{8,0,1}, t_{9,2,3}, \{t_{10,4,5}, t_{10,14,15}\}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, \{t_{\underline{14},4,11}, t_{\underline{14},10,15}\},$
$\{t_{\underline{15},5,11}, t_{\underline{15},10,14}\}, \{t_{\underline{16},12,13}, t_{\underline{16},8,10}\}, \{t_{\underline{17},13,14}, t_{\underline{17},9,15}\}.$

The extended graph

G_{s_0} : $t_{8,0,1}, t_{9,2,3}, \boldsymbol{t_{10,4,5}}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, \boldsymbol{t_{\underline{14},4,11}}, \boldsymbol{t_{\underline{15},5,11}}, \boldsymbol{t_{\underline{16},12,13}}, \boldsymbol{t_{\underline{17},13,14}},$
G_{s_1} : $t_{8,0,1}, t_{9,2,3}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, \boldsymbol{t_{\underline{14},4,11}}, \boldsymbol{t_{\underline{15},5,11}}, \boldsymbol{t_{10,14,15}}, \boldsymbol{t_{\underline{16},12,13}}, \boldsymbol{t_{\underline{17},13,14}},$
G_{s_2} : $t_{8,0,1}, t_{9,2,3}, \boldsymbol{t_{10,4,5}}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, \boldsymbol{t_{\underline{15},5,11}}, \boldsymbol{t_{\underline{14},10,15}}, \boldsymbol{t_{\underline{16},12,13}}, \boldsymbol{t_{\underline{17},13,14}},$
... ...
$G_{s_{31}}$: $t_{8,0,1}, t_{9,2,3}, t_{11,6,7}, t_{\underline{12},8,9}, t_{13,9,10}, \boldsymbol{t_{10,14,15}}, \boldsymbol{t_{\underline{14},10,15}}, \boldsymbol{t_{\underline{15},10,14}}, \boldsymbol{t_{\underline{16},8,10}}, \boldsymbol{t_{\underline{17},9,15}}.$

32 single graphs

Further Optimization

Removing Redundant Nodes

After splitting the extended graph, the nodes in different single graphs may have different in-degrees and out-degrees.
The out-degree 0 means that the node is not used to generate other nodes.

Property

Given a DAG, if $\text{out}(u) = 0$, u must be the target node or the redundant node.

32 graphs: 18 with 10 XORs, 12 with 9 XORs, 2 with 8 XORs.

Further Optimization

Removing Redundant Nodes

After splitting the extended graph, the nodes in different single graphs may have different in-degrees and out-degrees.
The out-degree 0 means that the node is not used to generate other nodes.

Property

Given a DAG, if $\mathrm{out}(u) = 0$, u must be the target node or the redundant node.

32 graphs: 18 with 10 XORs, 12 with 9 XORs, 2 with 8 XORs.

Further Optimization

Removing Redundant Nodes

After splitting the extended graph, the nodes in different single graphs may have different in-degrees and out-degrees.
The out-degree 0 means that the node is not used to generate other nodes.

Property

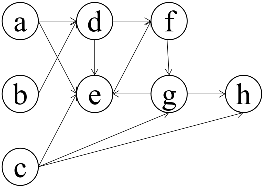
Given a DAG, if $\mathrm{out}(u) = 0$, u must be the target node or the redundant node.

32 graphs: 18 with 10 XORs, 12 with 9 XORs, 2 with 8 XORs.

Final Results

Wrong Graph

The wrong graph is an incorrect circuit for the corresponding matrix, which usually contains the cycles in the graph. Unit nodes cannot generate the nodes in the cycle.



Reduced graph 1	$t_{8,0,1}, t_{9,2,3}, \boldsymbol{t}_{\mathbf{10,4,5}}, t_{11,6,7}, t_{\underline{12},8,9}, \boldsymbol{t}_{\underline{\mathbf{14}},\mathbf{4,11}}, \boldsymbol{t}_{\underline{\mathbf{15}},\mathbf{5,11}}, \boldsymbol{t}_{\underline{\mathbf{16}},\mathbf{8,10}}, \boldsymbol{t}_{\underline{\mathbf{17}},\mathbf{9,15}},$
Reduced graph 2	$t_{8,0,1}, t_{9,2,3}, t_{11,6,7}, t_{\underline{12},8,9}, \boldsymbol{t}_{\underline{\mathbf{14}},\mathbf{4,11}}, \boldsymbol{t}_{\underline{\mathbf{15}},\mathbf{5,11}}, \boldsymbol{t}_{\mathbf{10,14,15}}, \boldsymbol{t}_{\underline{\mathbf{16}},\mathbf{8,10}}, \boldsymbol{t}_{\underline{\mathbf{17}},\mathbf{9,15}},$
Reduced graph 3	$t_{8,0,1}, t_{9,2,3}, \boldsymbol{t}_{\mathbf{10,4,5}}, t_{11,6,7}, t_{\underline{12},8,9}, \boldsymbol{t}_{\underline{\mathbf{15}},\mathbf{5,11}}, \boldsymbol{t}_{\underline{\mathbf{14}},\mathbf{10,15}}, \boldsymbol{t}_{\underline{\mathbf{16}},\mathbf{8,10}}, \boldsymbol{t}_{\underline{\mathbf{17}},\mathbf{9,15}},$
Reduced graph 4	$t_{8,0,1}, t_{9,2,3}, \boldsymbol{t}_{\mathbf{10,4,5}}, t_{11,6,7}, t_{\underline{12},8,9}, \boldsymbol{t}_{\underline{\mathbf{14}},\mathbf{4,11}}, \boldsymbol{t}_{\underline{\mathbf{15}},\mathbf{10,14}}, \boldsymbol{t}_{\underline{\mathbf{16}},\mathbf{8,10}}, \boldsymbol{t}_{\underline{\mathbf{17}},\mathbf{9,15}}.$

Algorithm

Algorithm 2 ExtendGraph2()

Input: A single graph G_s
Output: The set $\mathcal{G}_2$ containing all the reduced graphs

$G_e = \text{GenerateExtendedGraph}(G_s, 2)$ ▷ The extended graph
$\mathcal{G}_2 = \text{SplitExtendedGraph}(G_e)$ ▷ Generating the single graphs
for each $G_r \in \mathcal{G}_2$ **do** ▷ Removing additional nodes
    $G_r = \text{RemovingRedundantNodes}(G_r)$
end for
for each $G_r \in \mathcal{G}_2$ **do** ▷ Deleting wrong graphs
    if $\text{TopologicalOrdering}(G_r) = \text{error}$ **then**
        $\mathcal{G}_2 \leftarrow \mathcal{G}_2 / \{G_r\}$
    end if
end for
return $\mathcal{G}_2$

1. Generate the extended graph.
2. Split the extended graph.
3. Remove redundant nodes.
4. Delete wrong graphs.

Outline

Transforming Framework

Limitation

If we use g_ϵ-XOR metric, only i-input $(i \leq \epsilon + 1)$ xor gates can be used.

Transformation

$\lambda_1 e_1 + \lambda_2 e_2 + ... + \lambda_{n-1} e_{n-1}$ to $\lambda_1 e_1' + \lambda_2 e_2' + ... + \lambda_{n-1} e_{n-1}' + \lambda_n e_n'$

Transforming Framework

Question 1

Which node can be removed?

Question 2

How to remove one node?

Question 3

Which node should be removed first when both nodes can be removed?

Question 1: Which Node can be Removed?

Proposition

Suppose that the circuit is with the g_ϵ-XOR metric and $in(u) = j$. Only when the in-degree k of every node in $O(u)$ is not greater than $\epsilon + 2 - j$, can we remove u.

u can be removed.

Question 2: How to Remove One Node?

Proposition

Let N be the maximum value such that $(N+1)\lambda_1 - N\lambda_2 > 0$ holds. Given a circuit with XOR2 gates, it can reduce the cost by removing the nodes with out-degree n $(n \leq N)$.

$$u = a \oplus b$$
$$v = u \oplus c$$
$$w = u \oplus d$$

Condition: $3\lambda_1 - 2\lambda_2 > 0$.

$$v = a \oplus b \oplus c$$
$$w = a \oplus b \oplus d$$

The area of the new circuit is $2\lambda_2 < 3\lambda_1$.

Question 3: Which Node should be Removed first

Proposition

Suppose that the upper bound is N. If $\mathrm{out}(u) = m$ and $\mathrm{out}(v) = n$ ($n < m \leq N$), removing v will reduce more cost than u.

The out-degree of u is m.

The out-degree of v is n.

We have $m > n$.

We remove v first.

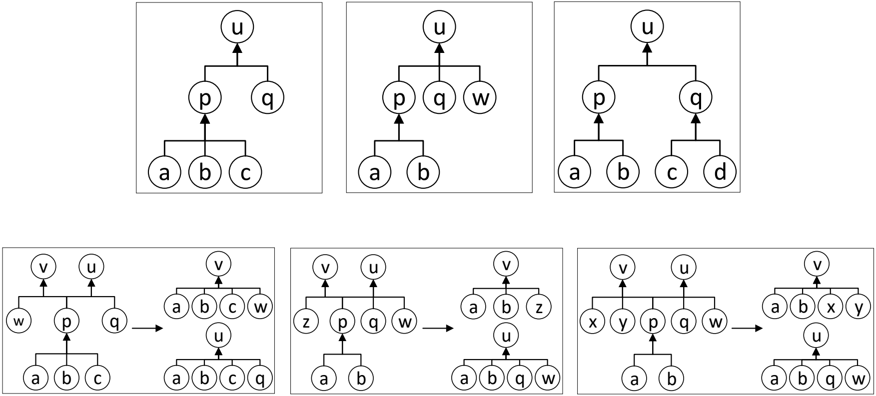
3-Input XOR Gate

Algorithm 3 EGT2()

Input: A single graph G_s
Output: A set $\mathcal{G}_3$ containing all the reduced graphs with 2/3-input xor gates

```
  G_2 = ExtendGraph2(G_s)            ▷ Containing the reduced graphs with XOR2 gates
  N ← 0                                                      ▷ The upper bound
  while ((N + 1) + 1)λ_1 − (N + 1)λ_2 > 0 do
      N ← N + 1
  end while
  for each G_r ∈ G_2 do                                      ▷ Removing nodes
      T = TopologicalOrdering(G_r)
      The set U containing all the target nodes in G_r
      n ← 1
      while n ≤ N do
          for each node u in T do
              if u ∉ U, out(u) = n, and O(u) ∩ U = ϕ then
                  We remove u, delete corresponding edges, and add n operations in G_r
                  Put the nodes in O(u) into U
              end if
          end for
          n ← n + 1
      end while
      G_3 ← G_3 ∪ {G_r}
  end for
  return G_3
```

4-Input XOR Gate



4-Input XOR Gate

Algorithm 4 EGT3()

Input: A single graph G_s
Output: A set $\mathcal{G}_4$ containing all the reduced graphs with 2/3/4-input gates

```
G_4 ← φ
G_3 ← φ
G_2 = EGT2(G_s)
for each graph G_r in G_2 do
    G' = EGT3(G_r)
    for each graph G in G' do
        G_3 ← G_3 ∪ {G}
    end for
end for
N_1 ← 0
N_2 ← 0
while λ_2 + (N_1 + 1)(λ_1 − λ_3) > 0 do
    N_1 ← N_1 + 1
end while
while λ_1 + λ_2 − λ_3 − N_2 · min((λ_2 − λ_1), (λ_3 − λ_2)) > 0 do
    N_2 ← N_2 + 1
end while
for each G_r ∈ G_3 do                                          ▷ Removing nodes
    T = TopologicalOrdering(G_r)
    The set U containing all the target nodes in G_r
    n_1, n_2 ← 1
    while n_1 ≤ N_1 or n_2 ≤ N_2 do
        for each node u in T do
            if n_1 ≤ N_1, u matches Type 1, O(u) ∩ U = φ, and u ∉ U then
                Remove u, delete corresponding edges, and add edges from I(u) to O(u)
                Put the nodes in O(u) into U
            end if
            if n_2 ≤ N_2, u matches Type 2, O(u) ∩ U = φ, and u ∉ U then
                Remove u, delete corresponding edges, and add edges from I(u) to O(u)
                Put the nodes in O(u) into U
            end if
        end for
        n_1 ← n_1 + 1
        n_2 ← n_2 + 1
    end while
    G_4 ← G_4 ∪ {G_4}
end for
return G_4
```

Outline

Application

Matrix	XZLBZ[a]	[BDK+21][b]	[BFI21][b]	XZLBZ+BFI[b]	**XZLBZ+EGT2**[b]	**XZLBZ+EGT3**[c]
AES [DR20]	306.3 (92)	258.9 (12, 47)	260.3 (39, 28)	259.0 (26, 37)	**255.0 (29, 34)**	**243.0 (22, 21, 12)**
ANUBIS [BR00]	329.6 (99)	274.2 (11, 51)	293.0 (60, 20)	**270.3 (35, 33)**	**270.3 (35, 33)**	**253.6 (21, 24, 12)**
CLEFIA M_0 [SSA+07]	326.3 (98)	271.63 (13, 49)	293.0 (60, 20)	276.3 (34, 35)	**270.9 (31, 36)**	**258.9 (23, 16, 18)**
CLEFIA M_1 [SSA+07]	342.9 (103)	298.9 (3, 62)	294.3 (38, 36)	292.9 (39, 35)	**283.6 (32, 38)**	**270.2 (20, 27, 13)**
FOX MU4 [JV04]	452.8 (136)	-	**353.5 (46, 43)**	374.2 (48, 46)	372.2 (46, 47)	**347.5 (32, 26, 20)**
JOLTIK [JNP15]	146.5 (44)	**122.5 (6, 22)**	127.8 (16, 16)	126.5 (10, 20)	123.8 (12, 18)	**115.8 (9, 12, 5)**
MIDORI [BBI+15]	79.9 (24)	74.5 (0, 16)	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4, 0)**
PRINCE M_0, M_1 [BCG+12]	79.9 (24)	74.5 (0, 16)	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4, 0)**
PRIDE $L_0 - L_3$ [ADK+14]	79.9 (24)	74.5 (0, 16)	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4, 0)**
QARMA128 [Ava17]	159.8 (48)	-	145.8 (34, 7)	145.8 (34, 7)	**144.5 (28, 11)**	**144.5 (28, 11, 0)**
QARMA64 [Ava17]	79.9 (24)	74.5 (0, 16)	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4)**	**71.9 (16, 4, 0)**
SMALLSCALE AES [CMR05]	143.1 (43)	**111.8 (0, 24)**	123.8 (19, 13)	123.8 (19, 13)	121.8 (17, 14)	118.4 (5, 9, 10)
TWOFISH [SKW+98]	369.6 (111)	317.5 (17, 56)	338.9 (43, 42)	312.9 (31, 45)	**306.9 (25, 48)**	**293.5 (13, 28, 20)**

[a] Using 2-input xor gates.
[b] Using 2/3-input xor gates.
[c] Using 2/3/4-input xor gates.

Conclusion

- The transforming framework
- The graph extending algorithm

Thanks for Your Attention!